Reliability-Aware Runtime Power Management for Many-Core Systems in the Dark Silicon Era

Amir M. Rahmani, *Member, IEEE*, Mohammad-Hashem Haghbayan, *Student Member, IEEE*, Antonio Miele, *Member, IEEE*, Pasi Liljeberg, *Member, IEEE*, Axel Jantsch, *Member, IEEE*, and Hannu Tenhunen, *Member, IEEE*

***Abstract*— Power management of networked many-core systems with runtime application mapping becomes more challenging in the dark silicon era. It necessitates considering network characteristics at runtime to achieve better performance while honoring the peak power upper bound. On the other hand, power management has a direct effect on chip temperature, which is the main driver of the aging effects. Therefore, alongside performance fulfillment, the controlling mechanism must also consider the current cores' reliability in its actuator manipulation to enhance the overall system lifetime in the long term. In this paper, we propose a multiobjective dynamic power management technique that uses current power consumption and other network characteristics including the reliability of the cores as the feedback while utilizing fine-grained voltage and frequency scaling and per-core power gating as the actuators. In addition, disturbance rejecter and reliability balancer are designed to help the controller to better smooth power consumption in the short term and reliability in the long term, respectively. Simulations of dynamic workloads and mixed criticality application profiles show that our method not only is effective in honoring the power budget while considerably boosting the system throughput, but also increases the overall system lifetime by minimizing aging effects by means of power consumption balancing.**

***Index Terms*— Dark silicon, feedback controller, lifetime reliability, networks on chip (NoCs), power management, runtime mapping.**

I. Introduction

THE number of transistors per chip still steadily increases by about 2.0× for each technology node generation. However, due to increased power densities and consequent thermal issues, power budget represents a constraint in the usability of such computational resources: in fact, the number of transistors that can be used at the same time on a chip increases with a lower trend (only 1.4× each generation [1]). Consequently, only a subset of the available processing cores can be used at full throttle at any time, while the rest of the resources have to be left inactive, thus representing dark silicon [2], [3]. Thermal issues do represent a problem not only in the short term by idling resources but also in the long term. In fact, as discussed in the ITRS reports in 2011 [4], due to transistor shrinking the higher temperatures cause the devices to be more susceptible to aging and wear-out phenomena (such as time-dependent dielectric breakdown, thermal cycling, and electromigration), thus decreasing the system lifetime. As stated in [5], many failure mechanisms are exponentially dependent on temperature, and a 10 °C–15 °C difference in operating temperature may result in a 2× difference in the overall lifespan of a device.

An investigated solution to mitigate dark silicon phenomenon is the near-threshold computing (NTC), also dubbed as dim silicon [6], and more in general acting on the voltage/frequency (VF) levels of the processing cores. The basic idea is to increase the number of concurrently active cores by considerably lowering their operating VF level. However, to implement an efficient NTC-based approach, an intelligent and stable power management mechanism is required. Such a mechanism becomes more challenging, considering that current and future many-core systems integrate various hundreds of networked resources and use to execute a highly variable and unpredictable workload. Therefore, to accurately handle an upper limit on power consumption (fixed Thermal Design Power, TDP, or dynamic Thermal Safe Power, TSP [7]), and at the same time guarantee a certain level of quality of service (QoS) in the running applications, a feedback control mechanism monitoring several parameters of the system is mandatory. Finally, the quest of such a feedback control mechanism is even more motivated when we aim also at limiting the aging trend and to prolong system lifetime.

The previous work on feedback-based dynamic power management (DPM) for multicore/many-core systems can be classified into two main categories.

1) Techniques that use workload and network characteristics as feedback (e.g., queue utilization and injection rate) and then adjust VF of processing elements, routers, or VF islands (VFIs) accordingly (see [8], [9]).
2) Power budgeting (i.e., capping) techniques [10], [11] that utilize chip/per-core power measurement and per-core performance counters (i.e., core utilization) as feedback, and then apply dynamic voltage and frequency

Manuscript received March 7, 2016; revised May 27, 2016; accepted July 7, 2016. This work was supported by the Academy of Finland project entitled "MANAGE: Data Management of 3D Systems for the Dark Silicon Age," University of Turku Graduate School (UTUGS).

A. M. Rahmani, M. H. Haghbayan, and P. Liljeberg are with the Department of Information Technology, University of Turku, Turku 20520, Finland (e-mail: amirah@utu.fi; mohhag@utu.fi; pasi.liljeberg@utu.fi).

A. Miele is with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan 20133, Italy (e-mail: antonio.miele@polimi.it).

A. Jantsch is with Vienna University of Technology, Vienna 1040, Austria (e-mail: axel.jantsch@tuwien.ac.at).

H. Tenhunen is with the Department of Information Technology, University of Turku, Turku 20520, Finland and also with the Department of Industrial and Medical Electronics, KTH Royal Institute of Technology, Stockholm 16440, Sweden (e-mail: hannu@kth.se).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2016.2591798

scaling (DVFS) or per-core power gating (PCPG) techniques to optimize the system performance within a fixed power cap (i.e., TDP).

Even though all the techniques in these categories efficiently save and control the power consumption for their target platforms, they are not comprehensive and do not take aging effects into account. The techniques in the first category do not consider any safe upper bound on the total system power consumption (i.e., TDP) at runtime, and therefore, they do not feed any power metric back to the management unit. The power capping techniques from the second category are mainly targeted for bus-based systems. Therefore, they are unable to address the power management issues in many-core systems that are typically based on a network on chip (NoC) and with multiple applications running simultaneously. Finally, very few approaches in the literature also consider aging issues in the dynamic mapping decisions, with [11] as a notable exception. In conclusion, we contend that dark silicon awareness necessitates an efficient multiobjective feedback-based control approach that considers many parameters such as workload characteristics, per-core power, aging and performance measurements, network-load, and total chip power consumption all together.

We propose a novel comprehensive dark-silicon-aware and reliability-aware power management framework for NoC-based many-core systems under limited power budget (both TDP and TSP) and running dynamic workloads, by supporting runtime mapping. This framework benefits from a multiobjective feedback-based controller that performs the following:

1) pursues performance and lifetime reliability optimization while fulfilling the power budget;
2) acts on PCPG and DVFS;
3) considers several monitoring parameters such as workload characteristics, network congestion, and power/performance/aging characteristics of processing cores.

A preliminary version of the approach has been proposed in [12]. We extend it to consider lifetime reliability issues together with power and performance optimization as follows.

1) Adding the reliability analysis unit to calculate fine grained reliability based on the temperature feedback profile from the system.
2) Developing novel decision policies targeted for two different operating modes: overboosting mode, when the system is experiencing an intensive workload, and reliability-aware mode, when the nonintensive workload offers the controller the opportunity to prolong the system lifetime.
3) Extending the metrics to performing VF scaling to consider reliability of the system.
4) Adding an additional reliability balancing module running at coarse time intervals.
5) Evaluating the efficiency of our approach to provide high performance while prolonging the system's lifetime and fulfilling the given power budget.

The rest of this paper is organized as follows. In Section II, related work is presented. An overall view of the NoC-based many-core system together with the envisioned runtime management framework is presented in Section III, while subsequent sections, Sections IV and V, present the internals of the companion monitoring infrastructure and the proposed power controller, respectively. An experimental evaluation of the proposed approach is provided in Section VI, and finally, Section VII draws conclusions.

II. Related Work

Various approaches for dark-silicon-aware power management have been proposed in the literature. In [10], a hierarchical power management controller for asymmetric multicores is demonstrated in an ARM big.LITTLE platform. Ma and Wang [11] propose a similar technique called *PGCapping* by exploiting power gating and DVFS for power capping in symmetric multicore processors. These two approaches use a feedback loop to manage power consumption, and their main limitation is that they target a bus-based architecture. Therefore, they do not consider many relevant issues manifesting in NoC-based many-core systems. Khan *et al.* [13] propose a method to allocate shared energy-efficient accelerators among competing applications to minimize the overall power while fulfilling the application deadlines. In [14], a hierarchical technique is proposed to partition TDP among running applications. Shafique *et al.* [15] propose a power management technique for many-core systems. In their platform, applications use feedback to monitor other running application activity to self-manage their power consumption and degree of parallelism. However, none of these works consider any power feedback from the system at runtime and instead use a predefined dedicated power budget.

Chen and Marculescu [16] present a power allocation technique for many-core system performance improvement under power constraints. They formulate a performance optimization problem and apply an optimal power allocation method using on-line distributed reinforcement learning. This contribution does not consider on-chip communication. However, it should be noted that this work is orthogonal to our multiobjective control management and can complement and enhance our method by integrating the concept of on-line learning toward more efficient global power budget reallocation. In [8], a control-based approach is proposed to minimize dynamic power in multi-core device made of multiple VFIs. Their goal is to determine optimal operating frequencies for both cores and routers. This work is not dark silicon aware either, as they do not utilize feedback from power sensors to avoid violating the TDP/TSP.

Haghbayan *et al.* [17] present a power management technique for many-core systems using power feedback from the system to meet the TDP bound. This technique is also categorized into the class of single objective control approaches as it lacks feedbacks from workload characteristics and per-core performance measurements from the system during DVFS. In addition, this technique is designed for fixed TDP and does not benefit from a dedicated disturbance rejector to handle sudden overshoots when new applications start.

A more advanced approach for power management of many-core systems in the dark silicon era has been proposed in [12].

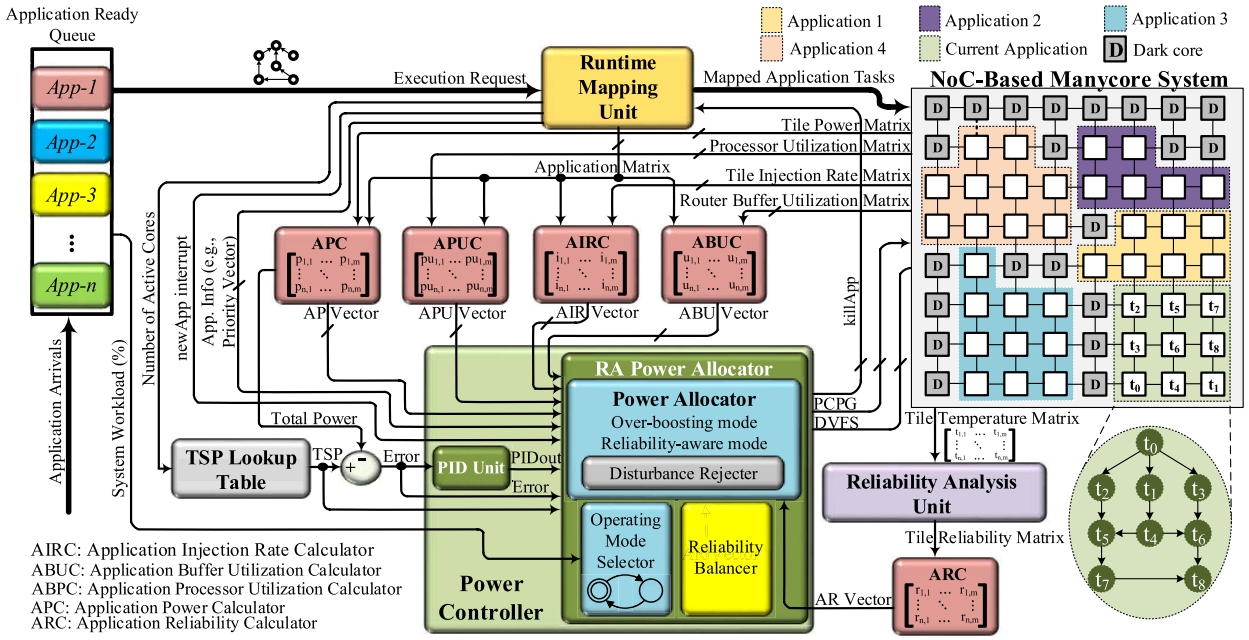

Fig. 1. Overview of our multiobjective dark-silicon-aware power management system.

The controller implements a feedback control loop collecting a large set of information from the running workload and the underlying architecture (such as network congestion and performance/power consumption status of the various cores) and acting on both PCPG and DVFS to optimize system performance while fulfilling the available power budget.

With the exception of the bus-based power capping technique presented in [11], none of the above approaches take into account the effects on the thermal stress with its impact on aging. As a matter of fact, reliability management cannot be performed by only considering TDP or TSP; in fact, TDP/TSP can only guarantee that excessive temperature peaks are not reached but does not mitigate the effects of high temperatures in the long term. For this reason, some approaches [18], [19] act on DVFS to balance the tradeoff among performance, power consumption, and reliability. However, they are mainly targeted for bus-based systems.

III. System Overview

Fig. 1 shows the overall architecture of the considered system. It consists of a classical hardware many-core platform and the above software stack, composed of the newly proposed runtime resource management layer, and the running application workload. The hardware platform we consider captures the main characteristics of modern many-core systems, such as the Intel single-chip cloud computer (SCC [20]) or the Kalray MPPA [21]. This platform contains a set of homogeneous processing tiles organized in a 2-D mesh-based topology and connected by an NoC. Moreover, the platform is connected to a host machine controlling all the activities.

Each tile contains a processing core provided with a local private instruction/data memory and a communication controller connecting to a corresponding router in the NoC; communications among tiles are performed by means of a message-passing protocol over the NoC. Finally, we assume each tile to be provided with HW sensors for temperature and power consumption measurements (as in [16], [20], and [11]), performance counters measuring the number of executed instructions (as in [20]), and actuators for PCPG and DVFS (as in [11] and [16]).

The many-core system generally executes data-intensive and highly parallel applications, organized in a group of interdependent processing tasks. Applications may also present priority requirements (e.g., soft realtime and non-realtime) correlating to the level of expected QoS. Such requirements are here expressed in terms of a priority index that can be used to rank applications while the QoS in terms of a minimum throughput (or a deadline on the overall execution time) to be guaranteed. To be executed, each application needs to be mapped on the architecture, by reserving a group of tiles on which the application tasks are dispatched.

The system supports the concurrent execution of a highly dynamic workload, characterizing the nowadays on-demand computing scenario. Multiple applications dynamically arrive with an unknown trend and are characterized by different structures (in terms of the number of tasks and their organization), latencies, and QoS requirements. Therefore, the architecture loads a software layer for runtime resource management performing two main tasks: runtime mapping and DPM. More precisely, similar to Intel SCC or Kalray MPPA, the main part of this software layer, implementing all the decision policies for mapping and power management, is executed on the host machine. Then, each tile loads a minimal scheduler running the received tasks and managing transmissions. Furthermore, the tile scheduler also executes a set of routines performing monitoring activities by means of the available sensors.

The runtime mapping unit (RMU) is in charge of performing the dispatching of the workload. Entering applications are initially stored in a ready queue, and the RMU performs the mapping of a ready application according to the availability of idle resources (i.e., the tiles set as dark since they are

not allocated for the execution of any other applications) and power assigned by the power controller. The RMU also categorizes information of the applications running on the system, defined as running application information (RAI), such as the idle/active tile matrix, the application mapping matrix, and the applications' priority list.

The second component within the runtime resource management layer is the power controller. This module is in charge of accurately handling the available power budget and distributing it among the various groups of tiles executing the running applications. The available power budget is used as a target value for the power controller unit, as shown in Fig. 1, and can be TDP or TSP (only TSP is mentioned in the figure). The activity of the controller is supported by means of a set of auxiliary monitoring modules preprocessing RAI gathered from the RMU and raw measures from the architectural sensors and providing a set of high-level information and statistics driving the decisions on DVFS and PCPG actuation. It is worth noting that the power controller does not scale the voltage and frequency of on-chip interconnection network components (e.g., NoC routers and links), to ensure that there are no unnecessary delays in the transmission, thus causing additional static power consumption of waiting tile. Finally, the controller also provides commands to the RMU to coordinate mapping decisions with the power management.

The novel contribution of this paper is the power controller, with the utility modules, while a state-of-the-art RMU [22] will be considered. The monitoring and power management policies are presented in the following.

IV. System Monitoring

As depicted in Fig. 1, a set of auxiliary modules perform monitoring activities on the system to collect various kinds of information. Each module is implemented by means of two parts: 1) a software routines periodically running on each tile that collect data from HW sensors and send them to the host machine and 2) the counterpart function integrated in the software layer on the host machine that will group all values of the same type in a matrix-based data structure.

Many aspects in the current system status may be relevant to take the most efficient choices on DVFS and PCPG actuation in order to get high performance for each running application, and at the same time optimize the power consumption of the system and slowing down the aging trend of the processing units. As an example, it is intuitive that the change of the current VF level of a set of cores allocated to the execution of an application will have a direct effect on both the performance and the power consumption. Nevertheless, it is also fundamental to monitor the traffic on the NoC as congestion in the network may represent a bottleneck that would make the increase in VF on the tiles to have no gain on the performance, while adding an additional contribution on power consumption. Such a scenario becomes even more complex when considering that the proposed approach aims at also considering, as a novelty, system's lifetime reliability as the third objective. As a conclusion, the auxiliary modules collect detailed information on the status of the processing units, i.e., the tiles, and of the communication infrastructure at router level, and aggregate these data with the granularity of each single application.

A. Application-Level Power Calculator

This module periodically samples the tile's power consumption by means of the available HW sensors and sends it to the host machine. Then, the host machine groups all these values in a tile power matrix. According to the application matrix provided by the RMU, the application-level power calculator (APC) module calculates the current power consumption of each application by aggregating the values of all tiles involved in its execution creating the application power vector (APV), and transmits it to the power controller. As depicted in Fig. 1, the input matrices (as the ones of the subsequent modules), which store the collected tile-level measures, have the same shape and dimension of the architecture grid. Then, the output is a vector with a length equal to the number of running applications.

B. Application-Level Processor Utilization Calculator

As in [11], the APC exploits the performance counters integrated within each tile to collect the processor utilization matrix, which measures the utilization of each processing unit (in percentages) during the previous timing window. Then, the application-level processor utilization calculator (APUC) module aggregates the utilization values of the various tiles allocated for the same application by means of a sum. Such aggregated metrics, showing the performance demand of the running applications, form the application processor utilization vector ($APUV$), passed to the power controller.

C. Application-Level Packet Injection Rate Calculator

Another useful information to decide how to tune the VF level of the various tiles is related to the amount of traffic generated by each single application. As noted in [23], a communication-intensive application may not benefit from an increase in the VF level in terms of higher performance. Indeed, the network would represent a performance bottleneck and an increase in the VF level would only worsen such overall network congestion and consume more power without any benefit. Thus, we consider applications' network intensity in order to classify them into intensive and nonintensive ones, and we use application packets injection rate as a metric that closely correlates to network intensity.

The packets injection rate of each task is measured at the network interface of the allocated tile based on its recent history. The application-level packet injection rate calculator (AIRC) receives such measures in the form of a tile injection rate matrix, and it accordingly calculates the average injection rate for each application by selecting the related values according to the application matrix. The obtained application-level packet injection rate vector ($AIRV$) is transmitted to the power controller.

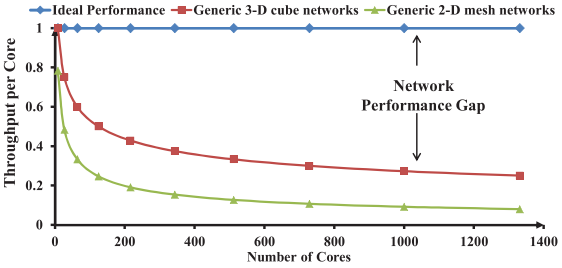

Fig. 2. Share of network bandwidth per core diminishes with the increasing number of cores as shown here for uniform traffic.

D. Application-Level Buffer Utilization Calculator

Monitoring the utilization level of the routers involved in the execution of a single application (i.e., the routers connected to the allocated tiles) is also fundamental to detect possible traffic congestions that would affect the application performance. Indeed, the network capacity does not grow proportionally to accommodate traffic generated with the increasing number of tiles [24]. As a consequence, the network traffic gets more easily congested proportionally with the number of tiles, and the overall throughput per tile decreases. Therefore, the total network performance gives a diminishing return due to increased communication distance. Fig. 2 shows that this issue leads to a network performance gap where every tile is not able to send or receive packets in every cycle.

In the considered platform, each router is equipped with a buffer utilization meter (as in [8]) monitoring router congestion levels in its recent history. More precisely, it dynamically measures the traffic in terms of average congestion level C_t at any given cycle t, by calculating the moving average of packet flow in every link of a router. Then, similar to the previous modules, the application-level buffer utilization calculator (ABUC) elaborates the router buffer utilization matrix and the application matrix to compute the application buffer utilization vector ($ABUV$) where buffer utilizations are averaged per application. As a final note, AIRC and ABUC are complementary in the analysis of the network status. In fact, the former monitors the amount of traffic generated by each single application, which depends on the characteristics of the application and the VF configuration of the processing cores, while the latter represents the actual congestion of the network due to the packets flow among the various routers, which depends on the intrinsic characteristics of the network in terms of the speed and routing scheme that would cause the actual accumulation of packets in the buffers.

E. TSP Lookup Table

The last input for the power controller is the available power budget. TDP has been classically used to define such a power budget at design time in order to avoid excessive temperatures. However, as shown in [7], TDP leads to a suboptimal usage of the device and results in large performance loss. For this reason, in [7], a dynamic power budgeting approach, called TSP, is defined as a function of the number of active (i.e., nondark) tiles in a system. In this paper, we use the TSP_{worst} strategy, which defines the safe power budget by considering the worst case mapping scenario (i.e., assuming that all the active tiles are physically packed and influence the temperature of their adjacent tiles). From a practical point of view, our approach is based on the precomputation of the TSP_{worst} values for a different number of active tiles in the system to build a TSP lookup table. Thus, at runtime, the *number of active tiles* can be used as an input of this lookup table to get the corresponding safe power budget P_{TSP}^{worst}. The TSP lookup table is queried whenever an application enters or leaves the system at runtime since these two events may cause a change in the number of active tiles. It should be noted that if using a fixed TDP value is desired, the lookup table can be simply replaced with the fixed value.

F. Reliability Monitor and Application-Level Reliability Calculator

The framework is also provided with a monitor devoted to the observation of the aging status of the various tiles. The most precise way to monitor the aging of each component is using hardware wear-out sensors. However, even though there is a large effort on the study and the design of wear-out sensors for the various aging phenomena (e.g., [25]), they are not currently integrated in commercial devices. For this reason, a commonly used strategy to estimate the aging status within a system is to employ statistical lifetime reliability models relying on the existence of per-core thermal sensors within the platform (e.g., [18]).

In this paper, we consider the standard lifetime reliability model for a given system based on a Weibull distribution [26]

$$R(t) = e^{-\left(\frac{t}{\alpha(T)}\right)^{\beta}} \tag{1}$$

where t is the current instant of time (generally measured in hours), T the constant worst-case processor temperature (Kelvin degrees), β the Weibull slope parameter, and $\alpha(T)$ the scale parameter or aging rate. The $\alpha(T)$ parameter formulation depends on the considered wear-out mechanisms, which are, for instance, electromigration, hot carrier injection, and thermal cycling.

Since the temperature of a tile varies in time due to the changes in the resource utilization and the intensity of the executed workload, the reliability monitor implements the advanced reliability model defined in [27] supporting temperature changes as follows:

$$R(t) = e^{-\left(\sum_{j=1}^{i} \frac{\tau_j}{\alpha_j(T)}\right)^{\beta}} \tag{2}$$

where τ_j represents the duration of each period of time with constant steady-state temperature T_j starting from time 0 up to time t (i.e., $t = \sum_{j=1}^{i} \tau_i$). From the implementation point of view, the module periodically samples the temperature T_i of each tile by means of the available sensor and computes the corresponding $\alpha_i(T)$. The reliability of the tile is updated by accumulating the duration of the last period τ_i and the computed $\alpha_i(T)$ to the exponent in (2). Finally, all reliability values are collected in a tile reliability matrix transmitted to the power controller. Then, a downstream module, called application-level reliability calculator, aggregates the information contained in the tile reliability matrix on the basis of

the application matrix to compute the so-called application reliability vector (ARV). Each value in the ARV is computed by multiplying the reliability values of the set of tiles allocated to a single application (i.e., from a reliability point of view, we consider the set of allocated tiles as a series system).

As a final note, the overhead caused by the proposed monitoring infrastructure is negligible in both computation and communication. In fact, the monitoring routines periodically collect few single floating point values and transmit them on the network. Moreover, at each period, the amount of traffic generated for power management is equal to approximately 1 kbit in the considered architecture in Fig. 1, which is mainly due to the transmission of the five floating point matrices.

V. Power Controller Unit

The novel power controller unit integrated in the overall system is shown in Fig. 1 (bottom). The unit is invoked periodically or at specific events, i.e., application's start and termination, and performs a DPM. The mission of this unit is to maximize the system throughput in terms of executed applications per unit of time and considering possible application QoS requirements, while not violating the available power budget, and at the same time to prolong the system lifetime by avoiding unnecessary thermal stress causing and slowing down the aging process in the various processing tiles. To this end, PCPG is used for power gating the cores that are not running any task, while an advanced tuning of per-core DVFS is performed to achieve the specified mission.

Within the power controller unit, a proportional–integral–derivative (PID) controller monitors the error between the actual power consumed by the system and the provided TSP over time. Then, the downstream reliability-aware power allocator unit uses the output of the PID controller together with the data collected from the monitoring units to handle the power management features. In particular, it tunes VF levels of the various tiles with the granularity of the mapped applications to optimize applications' execution times by exploiting available power budget in an efficient way. Moreover, in some specific situations where there is a deficit of power, the unit may also command the RMU to kill some running applications.

The reliability-aware power allocator is composed of three different submodules. An operating mode selector monitors the intensity of the current workload experienced by the system and properly selects the corresponding operating modes: overboosting mode when experiencing an intensive workload or reliability-aware mode in the other cases. Thus, the actual power allocator applies specific policies in the two operating modes; while in the first case the system is used at full speed to satisfy the highly demanding user requests, in the second case power allocation is performed by leveraging also the aging status of processing tiles. Finally, the last submodule, called the reliability balancer and invoked only in the reliability-aware mode, performs a minimal refinement of the decisions of the power allocator unit in order to further reduce the stress on the regions of the grid experiencing fast aging. The overall workflow of the reliability-aware power allocator is sketched in Algorithm 1, while the internals of the various modules are presented in the following sections.

Algorithm 1 Reliability-Aware Power Allocator

Inputs: $PID_{out}, RAI, ABUV, AIRV, APV, APUV, ARV, TSP,$
$newApp\_interrupt, Error, workload$
Output: $V_{PEs}, Freq_{PEs}, killApp$
Global Variables: $DVFSList, I_{set}, NI_{set}, C_{set}, NC_{set}, AppReliability_{set},$
$PCPowerLimit, timer, curr\_state$
Constant values: $bufferUtilizationLimit$, W_{th}, T_{th}
Body:
1: **if** $curr\_state = reliability\text{-}aware$ and $workload \geq W_{th} + \Delta$ **then**
2: $curr\_state \leftarrow over\text{-}boosting$;
3: **else if** $curr\_state = over\text{-}boosting$ and $workload \leq W_{th} - \Delta$ **then**
4: $curr\_state \leftarrow reliability\text{-}aware$;
5: $(V_{PEs}, Freq_{PEs}, killApp) \leftarrow$
$powerAllocator(PID_{out}, RAI, ABUV, AIRV, APV,$
$APUV, ARV, TSP, newApp\_interrupt, Error)$;
6: **if** $curr\_state = reliability\text{-}aware$ **then**
7: **if** $timer \geq T_{th}$ **then**
8: $(V_{PEs}, Freq_{PEs}) \leftarrow reliabilityBalnancing(ARV, RAI)$;
9: $timer \leftarrow 0$;
10: $timer \leftarrow timer + 1$;

A. PID Controller

A classical PID controller is used for controlling the power consumption of the system in a feedback control loop. In particular, the error between the current power consumption and the available TSP is monitored over time to decide the power budget that can be allocated to the current instant of time. The general formula for the PID controller is

$$PID_{out}(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt} \qquad (3)$$

where $PID_{out}(t)$, $e(t)$, K_p, K_i, and K_d are the controller output, error, proportional gain, integral gain, and derivative gain, respectively. The gains of the PID controller are appropriately adjusted after several MATLAB simulations. Two key factors were considered in our simulations, viz., the system stability and the system robustness against power disturbance. When considering a many-core system featuring runtime application mapping, there are three main events that influence the power trace curve and therefore $e(t)$:

1) the start of a new application;
2) the end of a running application;
3) the intra-application variations in the running applications due to the task dependencies and varying switching activities of the various tasks.

During our preliminary experimental analysis, we observed that the PID controller can efficiently support the power allocator in making decisions on handling the latter two situations. However, when an application enters the system, a high overshoot may easily happen especially when the application size is large and demands several dark cores to be activated. In fact, the newly incoming application will cause an increase in instantaneous power consumption. Since this phenomenon causes a drastic drop down in the error $e(t)$ not properly handled by the PID, we introduced a disturbance rejector unit within the power allocator to handle such power spikes in a proactive way (see Section V-C).

B. Operating Mode Selector

The two objectives the power controller unit aims at optimizing, i.e., the system's performance and reliability,

are frequently conflicting if not accurately handled. In fact, to achieve higher performance, it is necessary to increase VF and consume more power, but this would cause higher temperature hotspots within the chip and a faster aging. In contrast, a reliability-aware power capping approach would unstress the architecture but also worsen the performance. Another interesting difference between the two considered objectives is the highly different time horizon along which they are analyzed. Performance is generally measured in quite short time intervals lasting seconds to a few hours. In fact, applications arrive and leave the system with a very fast trend, and consequently, management decisions taken on each single application may considerably affect the system's performance. On the other hand, the aging status of a system is a very slow process, which is influenced by the average trend in the management decisions over a long period of time.

As discussed in [28] and [29], computing systems are generally subject to a variable workload over the time alternating periods of intensive activity and periods with a low number of running applications. Moreover, to fulfill the QoS demands of the running workload, it is not always necessary to execute at highest VF levels especially when the system is not experiencing an intensive workload composed of many applications in the ready queue. To face with this variability and handle the two conflicting goals, we define two different operating modes, called overboosting mode and reliability-aware mode, in which different decision policies are used in the power allocator. The *overboosting mode* is characterized by a highly intensive workload. For this reason, the system needs to work at full speed, and consequently, reliability issues are ignored. On the other hand, in the *reliability-aware mode*, metrics from the reliability monitoring are taken into account to avoid thermal hotspots while trying to obtain good performance as well. Therefore, the ultimate goal of these two operating modes is to obtain in the short-term optimal system's performance while mitigating unnecessary stress and wear out in the architecture in the long term. Therefore, the power controller unit is provided with an operating mode selector, which monitors the overall amount of workload the system is experiencing in the current period of time and consequently decides the operating mode to select. The amount of workload is computed by the operating mode selector in percentage value by monitoring the status of the ready queue. Then, the operating mode is transmitted to the power allocator that will use the corresponding decision policies. The operating mode selector internally behaves as a finite-state machine, as shown in Fig. 3, which on the basis of a given threshold switches between the overboosting mode and the reliability-aware mode. A tolerance guard band can be used around the threshold value to avoid excessive oscillations between the two different operating modes.

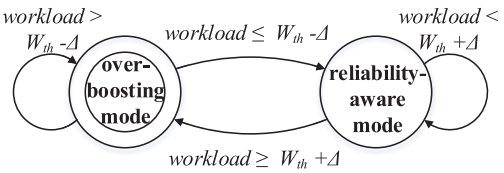

Fig. 3. State machine diagram of the operating mode selector.

Algorithm 2 Power Allocation Algorithm

Inputs: $PID_{out}, RAI, ABUV, AIRV, APV, APUV, TSP,$ $newApp\_interrupt, Error$
Output: $V_{PEs}, Freq_{PEs}, killApp$
Global Variables: $DVFSList, I_{set}, NI_{set}, C_{set}, NC_{set}, appReliability_{set},$ $PCPowerLimit$
Constant values: $bufferUtilizationLimit$
Body:
1: $PCPowerLimit \leftarrow \frac{TSP}{\#activeCores}$;
2: $(I_{set}, NI_{set}) \leftarrow IRClassifier(AIRV, RAI)$;
3: $(C_{set}, NC_{set}) \leftarrow BUClassifier(ABUV, RAI)$;
4: **if** $newApp\_interrupt = true$ **then**
5: $(V_{PEs}, Freq_{PEs}, killApp) \leftarrow$ $proactiveDistRej(error, RAI, ABUV, APV, APUV)$;
6: **else**
7: **if** $PID_{out} < 0$ **then**
8: $(V_{PEs}, Freq_{PEs}, killApp) \leftarrow$ $VF_{downscaler}(RAI, ABUV, APV, APUV, PID_{out},$ $PCPowerLimit)$;
9: **else**
10: $(V_{PEs}, Freq_{PEs}, killApp) \leftarrow$ $VF_{upscaler}(RAI, ABUV, APV, APUV, PID_{out},$ $PCPowerLimit)$;

C. Power Allocator

The overall behavior of power allocator is represented in Algorithm 2. As the first step, the module preprocesses some input data useful for subsequent elaborations. First, the per-core power limit *PCPowerLimit* is computed by dividing the TSP in the same portions by the number of active cores in the system (*#activeCores*). When TDP is chosen over TSP, this step can be ignored. Next, running applications are partitioned by means of the *IRClassifier* function in intensive (I_{set}) and nonintensive (NI_{set}) sets in terms of the network intensity, based on the $AIRV$. The classification algorithm has been borrowed from [23], where it is possible to find all the implementation details. Similarly, in the second step, applications are partitioned into congested (C_{set}) and noncongested (NC_{set}) sets, based on $ABUV$. In particular, as previously discussed, an application is tagged as congested if the average buffer utilization of its associated routers is larger than a predefined threshold (e.g., 75%). As a conclusion, each application is tagged with a 2-bit label that can get one of these values: $NI_$NC (nonintensive, noncongested), NI_C (nonintensive, congested), I_NC (intensive, noncongested), and I_C (intensive, congested). Applications are classified every time the controller is executed.

After classification, the event that has triggered the execution of the power controller and the current output of the PID controller are analyzed to select a proper DVFS policy. In particular, when the power controller is woken up by the periodic timer or by an application's completion event, the PID output, PID_{out}, is compared against zero. In the case of overshoot (i.e., when $PID_{out} > 0$, which means power consumption exceeding TSP/TDP), $VF_{downscaler}$ function is invoked to scale down VF levels of a subset of selected applications, while in the case of undershoot, the $VF_{upscaler}$ function performs VF upscaling. When a $newApp\_interrupt$ is asserted by the RMU due to the start of a new application, the disturbance rejecter module is invoked to proactively handle the large overshoot caused by the newly running

Algorithm 3 Voltage and Frequency Downscaling Function

Inputs: $RAI, ABUV, APV, APUV, PID_{out}, PCPowerLimit$
Outputs: $V_{PEs}, Freq_{PEs}, killApp$
Variables: $availableApps, targetApp, failedDVFS, appSet$
Body:

```
1: availableApps ← C_set ∪ NC_set;
2: while availableApps ≠ ∅ do
3:     targetApp ← ∅;
4:     appSet ← LP_apps(availableApps, RAI);
5:     if appSet ∩ C_set ≠ ∅ then
6:         appSet ← appSet ∩ C_set;
7:     else
8:         appSet ← appSet ∩ NC_set;
9:     if current_state = reliability-aware then
10:        targetApp ← lowReliability(appSet, appReliability_set);
11:    else
12:        targetApp ← lowD_prf−pwr(appSet, APV, APUV);
13:    (V_PEs, Freq_PEs, failedDVFS) ←
           DVFS(RAI, targetApp, PID_out, PCPowerLimit);
14:    if failedDVFS = true then
15:        availableApps ← availableApps \ {targetApp};
16:        if availableApps = ∅ then
17:            killApp ← targetApp;
18:            return;
19:    else
20:        DVFSList ← DVFSList ∪ {targetApp};
```

Algorithm 4 Voltage and Frequency Upscaling Function

Inputs: $RAI, ABUV, APV, APUV, PID_{out}, PCPowerLimit$
Outputs: $V_{PEs}, Freq_{PEs}, killApp$
Variables: $availableApps, targetApp, failedDVFS, appSet$
Body:

```
1: targetApp ← None;
2: availableApps ← DVFSList;
3: while targetApp = None do
4:     appSet ← availableApps ∩ NC_set ∩ NI_set;
5:     if appSet = ∅ then
6:         appSet ← availableApps ∩ NC_set ∩ I_set;
7:         if appSet = ∅ then
8:             appSet ← availableApps;
9:     appSet ← HP_apps(appSet, RAI);
10:    if current_state = reliability-aware then
11:        targetApp ← highReliablity(appSet, appReliability_set);
12:    else
13:        targetApp ← highD_prf−pwr(appSet, APV, APUV, PID_out);
14:    (V_PEs, Freq_PEs, failedDVFS) ←
           DVFS(RAI, targetApp, PID_out, PCPowerLimit);
15:    if failedDVFS = true then
16:        availableApps ← availableApps \ {targetApp};
17:        targetApp ← None;
18:        continue;
19: DVFSList ← DVFSList \ {targetApp};
```

application. $VF_{downscaler}$ and $VF_{upscaler}$ functions are discussed in the following.

1) $VF_{downscaler}$: Here, the main idea is to select the candidate applications to be downscaled among the ones with the lowest priority, letting the high-priority applications run at a higher QoS level. Algorithm 3 describes the VF downscaling process. The LP_{apps} function is applied on the overall set of running applications ($C_{set} \cup NC_{set}$) to get the subset of applications with the lowest priority, namely $appSet$. The filtering is performed according to the application priority vector contained in the RAI. Then, $appSet$ is even more filtered to consider only the subset of congested applications, with the aim at improving network throughput. In fact, tiles residing in a congested region of the architecture potentially dissipate unnecessary power (particularly static) due to low network throughput. As VF downscaling has also the effect on throttling of packet injection, it can alleviate the network congestion for such applications and save power. In case the selected $appSet$ is empty, congested set is replaced by the noncongested set (NC_{set}).

Finally, the target application to be downscaled is selected in the $appSet$ by means of a specific policy of the current operating mode. In the case of reliability-aware operating mode, the application characterized by the lowest reliability in the ARV is selected in order to reduce heating in the allocated tiles and, consequently, unstress the region. On the contrary, in the case of $overboosting$ mode, a $lowD_{prf\text{-}pwr}$ function is used to select the application with the lowest performance loss to power reduction ratio. The identified target application ($targetApp$) is then downscaled by the $DVFS$ function as per PID_{out} and $PCPowerLimit$.

The downscaling process continues iterating on the $availableApps$ list to look for the next target application. Moreover, the DVFS function will fail when it cannot throttle the target application any further according to the application type. This occurs when voltage and frequency cannot be reduced anymore. When the failed $DVFS$ variable is asserted, the application will be removed from $availableApps$ and the algorithm continues iterating on the list until an alternative application is found. Finally, if no application can be found to scale down, the last target application to be scaled down will be assigned to $killApp$, so that the RMU will kill it.

2) $VF_{upscaler}$: VF upscaling of processing tiles is presented in Algorithm 4. The algorithm works on the set of already downscaled applications, $DVFSList$, and performs a set of filtering to identify the target application. More precisely, first, $DVFSList$ is filtered to consider only the nonintensive and noncongested applications. In case the result is an empty set, the filtering is relaxed to consider only the nonintensive tag, and if also this last filter fails obtaining a nonempty set, the original $DVFSList$ content is considered. Finally, the last filtering is performed to select only the applications with the highest priority by means of the HP_{apps} function to consider the QoS demands. The basic idea at the basis of this selection process is that upscaling VF levels of a tile residing in a congested area and having a high injection rate may result in zero performance gain if on-chip communication network is the bottleneck. That is the reason why in VF upscaling process, in contrast with downscaling, a higher priority is given to congestion than application priority in the algorithm.

The target application to be upscaled is selected by means of a specific policy according to the current operating mode. In the case of *reliability-aware* operating mode, the application characterized by the highest reliability in the ARV is selected. In this way, we will put the stress in the youngest region of the device. On the contrary, in the case of $over\text{-}boosting$ mode, a $HighD_{prf\text{-}pwr}$ function is used to select the application with the highest performance loss to power increase ratio. The chosen target application is then upscaled by the $DVFS$ function as per PID_{out} and $PCPowerLimit$. The algorithm will continue iterating until an application is not found by removing the attempted candidates from the $availableApps$ list. Finally, the target application is removed from the $DVFSList$.

3) $HighD_{prf\text{-}pwr}$ and $LowD_{prf\text{-}pwr}$ Functions: These functions search for an application with the highest or lowest performance–power ratio, respectively, in a given set to be the target of VF upscaling or downscaling. The original idea of such functions can be found in [11], where the product

of core utilization ($Util$) and aggregated frequency ($Freq$) is used as a high-level computational capacity metric. In this metric, the frequency is weighted to deduct the idling cycles. We extend this metric by aggregating core utilization in an application (*appUtil*), provided by APUC, to calculate the performance of an application as

$$Perf_{current} = appUtil \times Freq_{current}. \tag{4}$$

Then, the performance–power ratio is calculated as

$$D_{prf-pwr} = \frac{Perf_{next} - Perf_{current}}{Power_{next} - Power_{current}} \tag{5}$$

where $Power_{current}$ is the power consumption of the current application provided by the APC unit and $Power_{next}$ and $Perf_{next}$ are the estimated power consumption and performance of the application after the DVFS process, respectively. The next VF level (V_{dd_next} and $Freq_{next}$) are estimated for the candidate applications based on the magnitude of PID_{out} and application size. $Perf_{next}$ and $Power_{next}$ are calculated as

$$Perf_{next} = Perf_{current} \times \frac{Freq_{next}}{Freq_{current}} \tag{6}$$

$$Power_{next} = Power_{current} \times \frac{Freq_{next}}{Freq_{current}} \times \left(\frac{V_{dd\_next}}{V_{dd\_current}}\right)^2. \tag{7}$$

After calculating $D_{prf-pwr}$ for all the applications in *appSet*, $lowD_{prf-pwr}$ and $highD_{prf-pwr}$ functions find the application with the lowest and highest $D_{prf-pwr}$ values as the target application for downscaling and upscaling, respectively.

4) DVFS Function: This function actuates the VF downscaling or upscaling of all the tiles allocated for the target application according to the specified PID_{out} and $PCPowerLimit$. In the case of downscaling, the minimum throughput specified for the application in the RAI is considered as an additional parameter to identify the minimum VF level applicable (as in [30]). Thus, the function identifies the minimum VF level for the downscaling or the maximum VF level for the upscaling to be applied to all the cores in order to satisfy the three constraints. If no solution can be found, the function returns a failure.

5) Proactive disturbance rejection: As discussed in the overall description of the approach, the commence of a new application may cause a drastic overshoot in the error between the consumed power and the available TSP/TDP and this phenomenon cannot be properly managed by the PID controller with the functionalities of the power allocator discussed so far. For this reason, these sporadic events can be efficiently managed by scaling down a selected set of already running applications in a proactive way in order to collect the power budget required by the new application before its start. The $proactiveDistRej$ function is depicted in Algorithm 5. The strategy first estimates the power consumption of the new application will have $appPredictedPower$, using the number of tasks of the application (N) and average power consumed by actively running tiles (P_{avg}), extracted from RAI and APV. Then, it computes $proactiveError$ as the difference between $Error$ and $appPredictedPower$. Actually, $proactiveError$ represents the PID input without the contribution of the new application supposing that it is already running. Actually, we need such a value since the aim is to collect the necessary power budget by scaling down other applications. Therefore, if $proactiveError$ is positive, there is availability of power budget for the new application, and it is mapped without any further scaling. Conversly, as shown in the second part of the algorithm, if $proactiveError$ is negative, other applications are scaled down. To perform this task, a proportional controller with gain K'_p is used. Here, the integral and derivative terms are removed because when such sporadic rises occur, history-based (i.e., integral term) or prediction-based (i.e., derivative term) decision making will most likely affect the controller's response. Then, the output of the controller (P_{out}) determines the extent to which currently running applications are to be scaled so that the new application can be mapped without violating TSP/TDP. Such downscaling of the currently running applications is performed as discussed above using the $VF_{downscaler}$ function.

Algorithm 5 Proactive Disturbance Rejection (*proactive DistRej()*)

Inputs: $Error, RAI, ABUV, APV, APUV, PCPowerLimit$
Outputs: $V_{PEs}, Freq_{PEs}, killApp$
Variables: $failedDVFS, appPredictedPower, proactiveError, P_{out}, P_{avg}$
Constant values: K'_p
Body:
1: $P_{avg} \leftarrow computeAvgPowerConsumption(RAI, APV)$;
2: $N \leftarrow countTaskOfNewApplication(RAI)$;
3: $appPredictedPower \leftarrow N \cdot P_{avg}$;
4: $proactiveError \leftarrow Error - appPredictedPower$;
5: **if** $proactiveError < 0$ **then**
6: $P_{out} \leftarrow K'_p \cdot proactiveError$;
7: $(V_{PEs}, Freq_{PEs}, killApp) \leftarrow VF_{downscaler}(RAI, ABUV, APV, APUV, P_{out}, PCPowerLimit)$;

D. Reliability Balancer

Even though when the controller is in the *reliability-aware* mode the amount of stress on the cores is used as one of the metrics in scale up/down process, there might be still possibility to further decrease the VF level of the cores with a negligible performance penalty. In this way, we can achieve a more efficient reliability balancing in the long term. For this, the last module in the power controller is a reliability balancer that tries to adjust the distribution of power on chip by marginal changes in VF levels of the tiles to reshape the unbalanced reliability distribution. Algorithm 6 shows a reliability balancing function that is running periodically whenever the system operating in the *reliability-aware* mode. In the first step, the average of all the application reliability values (ARV) is calculated by function $calcAvgRel$. After that, among all the running applications, if the reliability value for one application app_i is lower than the overall average reliability and this application can be downscaled according to its priority and required QoS (using the $check_{downscaling}$ function), the VF level of the allocated cores will be scaled down by one step. This downscaling provides a power slack (the difference between current power consumption and maximum upper bound). Hence, after that, one application among $DVFSList$ whose reliability is higher than $avgRel$ is selected to be

Algorithm 6 Reliability Balancing (*reliabilityBalancing()*)

Inputs: ARV, RAI
Outputs: V_{PEs}, $Freq_{PEs}$
Variables: $avgRel$, $usedAppls$
Body:
1: $avgRel \leftarrow calcAvgRel(ARV)$;
2: $usedAppls \leftarrow \emptyset$;
3: **for all** $app_i \in RAI$ **do**
4: **if** $appRel_i < avgRel$ and $app_i \notin usedAppls$ **then**
5: **if** $check_{downscaling}(V_{PEs}, Freq_{PEs}, app_i)$ **then**
6: **for all** $app_j \in DVFSList$ **do**
7: **if** $app_j \notin usedAppls$ and $i \neq j$ and $appRel_j > avgR$ **then**
8: $(V_{PEs}, Freq_{PEs}) \leftarrow VF_{onestep\_down}\ (app_i, RAI)$;
9: $(V_{PEs}, Freq_{PEs}) \leftarrow VF_{onestep\_up}\ (app_j, RAI)$;
10: $usedAppls \leftarrow usedAppls \cup \{app_i, app_j\}$;
11: break;

scaled one step up. The $reliabilityBalancing$ function is managed to be run at coarse interval time (see Line 10 in Algorithm 1) and its time interval is selected to be much longer than *RA-powerAllocator* to obviate considerable negative effect on the system performance that can be imposed by the $reliabilityBalancing$ function. It should be noted that as the sizes of the applications are different and the relationship between the VF level and power is nonlinear, $VF_{onestep\_down}$ scaling down the resources of an application and $VF_{onestep\_up}$ scaling up another application to adjust power might cause disturbance in overall power consumption. However, as such disturbance is marginal, it can be handled by the power allocator in the subsequent iterations.

VI. Experimental Evaluation

We experimentally evaluated the proposed multiobjective dark-silicon-aware power management approach by means of an in-house SystemC system-level many-core simulation framework, based on the one used in [12]. The tool is based on Noxim [31] to simulate the network infrastructure. A functional model of the processing tiles is implemented and characterized according to the specifications of the Niagara2 in-order core obtained from McPAT [32]. Physical scaling parameters were extracted from the Lumos framework [33] by considering the 16-nm CMOS technology node; in particular, we imported specifications for power modeling, voltage–frequency scaling, and TDP calculation. In the simulations, we considered a 12×12 many-core having a chip area equal to 138 mm^2. For the DVFS purpose, we use 15 VF levels (similar to Intel SCC) including near-threshold operation extracted from the Lumos framework (the *pessimistic* model). The minimum and maximum VF levels are set to (0.456 V, 300 MHz) and (0.908 V, 5.2 GHz), respectively. The frequency of the on-chip communication network (e.g., routers) is set to the maximum level (i.e., 5.2 GHz) to demonstrate that even at the maximum NoC speed, the network can get congested and should be taken into account in power management along with the other parameters. For the TSP calculation, we used the chip characterization in [7]. We set the ambient temperature to 45 °C, a threshold temperature that triggers thermal management to 80 °C, a maximum chip power consumption from the power supply to 300 W, and the power consumption of an inactive core to 0.3 W. Finally, we used Hotspot [34] to calculate the temperature from the power traces at runtime, and, for a proof of concept, for the reliability model, we considered the electromigration aging mechanism, characterized as in [27].

Two different application types have been defined: non-realtime, having a low priority, and soft realtime, with a high priority. Several instances of non-realtime applications spanning from 4 to 35 tasks have been generated using TGG [35]; communication and computation volumes are randomly distributed. We model MPEG4 and VOPD multimedia applications as soft realtime applications. We precalculate the minimum VF level for soft realtime tasks for their worst case contiguous mapping. The workload has been defined in terms of a random sequence of applications entering the incoming application ready queue. The workload mix consists of 30% of soft realtime applications and 70% of non-realtime ones. This sequence is kept fixed in all experiments for the sake of fair comparison.

The runtime resource management layer has been integrated into the described simulator. For the RMU, we adopted the state-of-the-art methods SHiC [22] and CoNA [36]. The controller period has been set to 50 ms as in [11].

In the first experimental campaign, we compare the power management capabilities and performance efficiency of the proposed approach by considering different approaches:

1) Our proposed reliability-aware multiobjective controller (*RA-MOC*).
2) Our multiobjective controller without reliability consideration (*MOC* as proposed in [12]).
3) *PGCapping* [11], where only the core's power–performance ratio is considered as feedback for the PCPG and per-core DVFS actuation.
4) *DSAPM* [17], where no information on performance and packet injection rate of tiles is used as feedback.
5) *without TSP/TDP constraint* where the system is not limited in terms of maximum power consumption.

The last one is the situation when, in reality, the chip would be damaged due to overheating. To perform a fair comparison, we aligned PGCapping and DSAPM techniques to the proposed approach, to use the same 15 VF levels for per-core DVFS. Finally, we consider a 10 s warm up phase for the results.

Fig. 4 presents the power traces of the system managed by the various considered approaches when honoring a constant TDP set to 126 W (the value has been calculated based on the chip power density). Deviation of power consumption from the TDP line reflects either violation or underutilization of power budget. It is worth mentioning that the power trace is reported after the warm up phase through which the system application entrance rate and power consumption become stable. It is possible to observe that PGCapping, DSAPM, and without-constraint power managements mostly tend to overshoot or undershoot from TDP presenting a considerable oscillation, while the proposed approach (with or without reliability management) is able to better exploit the available power budget. Even though PGCapping benefits from the cores' power–performance values fed back by the controller, and thus increases the system throughput to some extent, it suffers from the underutilization issue as it does not consider the network congestion and applications injection rates.

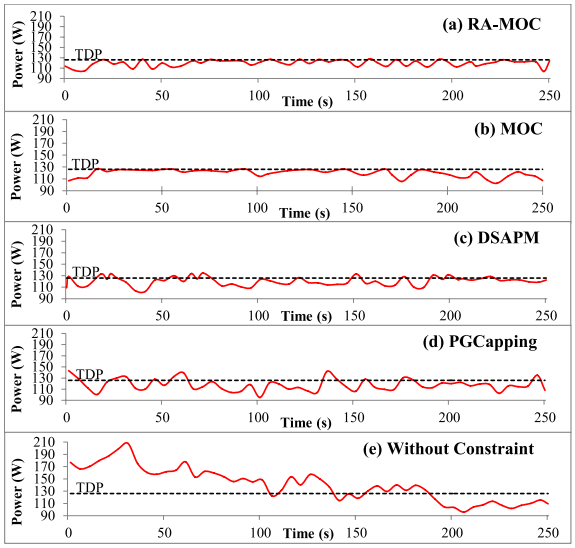

Fig. 4. System's power consumption to honor TDP.

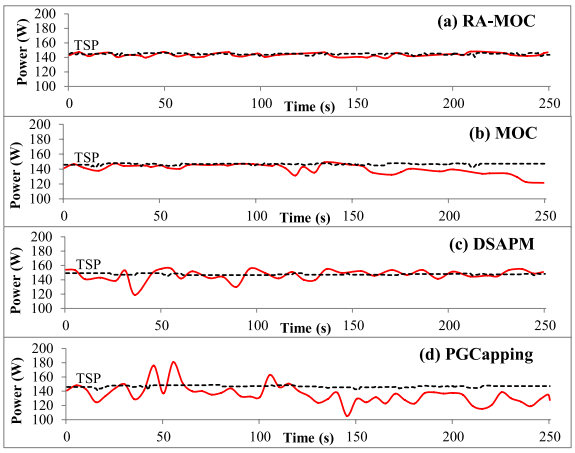

Fig. 5. System's power consumption to honor TSP.

DSAPM considers network congestion; however, it also suffers from the underutilization issue as it is agnostic of cores' performance value and applications' injection rates. Finally, both PGCapping and DSAPM techniques refuse to properly handle occasional overshoots due to new application arrivals. When considering the proposed approach, both MOC and RA-MOC have the best control on the power trace to stay in close proximity to TDP. In cases where power consumption exceeds TDP, the MOC controller rapidly reduces the power consumption by a proper VF scaling.

Fig. 5 demonstrates the aforementioned power management scenarios to honor dynamic TSP values. The conclusions we made for TDP are also valid for dynamic TSP; the MOC-based system is stable even when budget is changed at runtime. It is worth noting that TSP does not radically change (often between 141 W and 149 W) as the system is mostly busy and the majority of cores are active. To summarize, we computed the percentage of time the power budget is violated by the various power management policies over the overall simulation time. The results presented in Fig. 6 show that differently from the state-of-the-art approaches, the proposed MOC approach honors the TDP/TSP constraints for more than 99% of the simulation time.

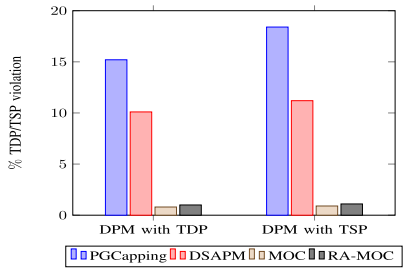

Fig. 6. TDP/TSP violation for different DPMs.

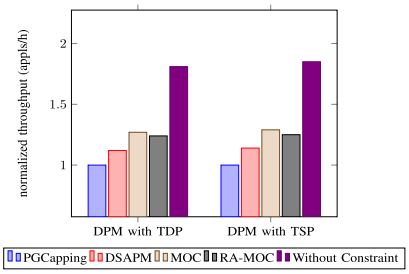

Fig. 7. Normalized throughput for different DPMs.

To assess the performance efficiency of our approach while optimizing the utilization of the power budget, we also analyzed in the same experiment the normalized throughput, in terms of the number of completed applications per unit of time, for the considered power management approaches. The results shown in Fig. 7 reveal that the proposed MOC method can significantly improve the overall system throughput for different power budget types (up to 29% compared with PGCapping and up to 15% compared with DSAPM). This result is obtained due to the advantage of our proposed multiobjective controller, which considers both the computation and communication aspects in power management. At the same time, RA-MOC has a slightly minor improvement, even if considerably better than state-of-the-art approaches. The motivation is related to the fact that it is also focused on avoiding excessive thermal stress and aging in the architecture, thus scarifying a bit on the performance. To show the impact of the power limit on the system performance, we also added the system throughput while no power management technique is applied (dubbed as *Without Constraint* in Fig. 7); as it can be seen, the dark silicon phenomenon has an impact of around 50% on the performance under the same workload.

In a second experimental campaign, we evaluate the reliability of the tiles during the time. To this purpose, we performed a long-term simulation in which we analyzed the evolution of the lifetime reliability of the various tiles for the overall operational life. To perform an accelerated experiment (with a reasonable simulation time), we enlarged the execution times of the applications to last a few days. Finally, to stimulate the system to switch with a realistic trend between the two different operating modes supported by the proposed approach, we modeled a variable workload during 24 h, according to the measures on a typical server in Facebook data center from [29]. In order to better show the efficiency of our proposed approach, we considered the proposed approach using only the reliability-aware power allocator technique without

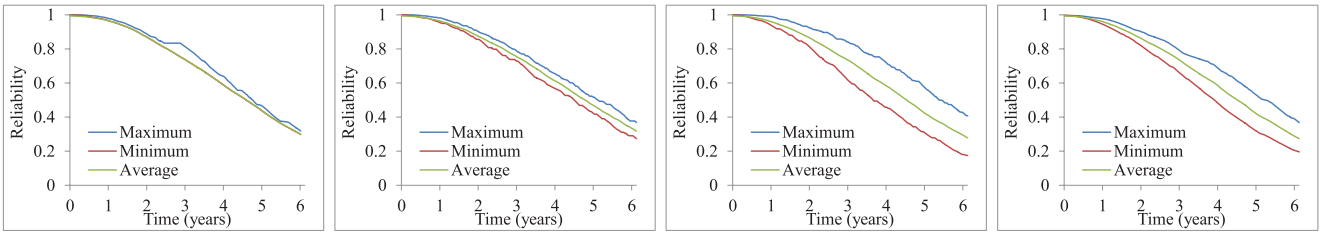

Fig. 8. Effect of reliability-aware power management approach on overall system reliability (TDP-based approach). (a) RA-MOC. (b) RA-MOC-no-balancing. (c) MOC. (d) PGCapping.

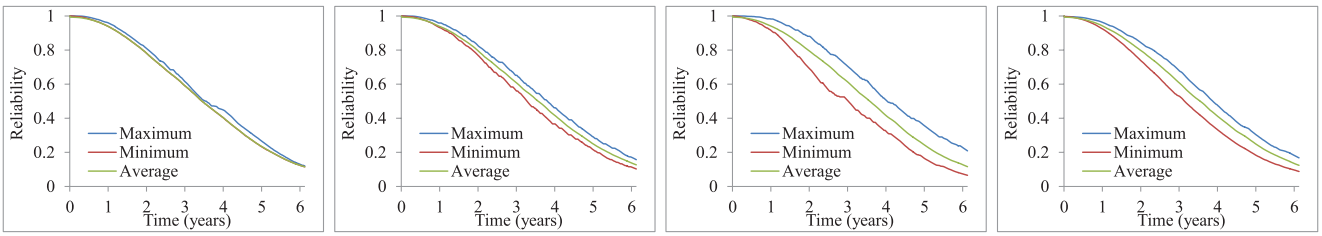

Fig. 9. Effect of reliability-aware power management approach on overall system reliability (TSP-based approach). (a) RA-MOC. (b) RA-MOC-no-balancing. (c) MOC. (d) PGCapping.

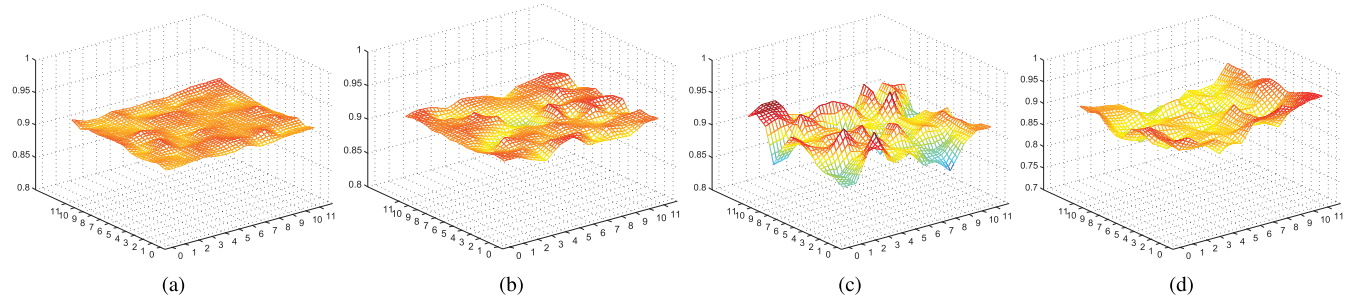

Fig. 10. Effect of reliability-aware power management approach on cores' reliability after 2 years of system activity (TDP-based approach). (a) RA-MOC. (b) RA-MOC-no-balancing. (c) MOC. (d) PGCapping.

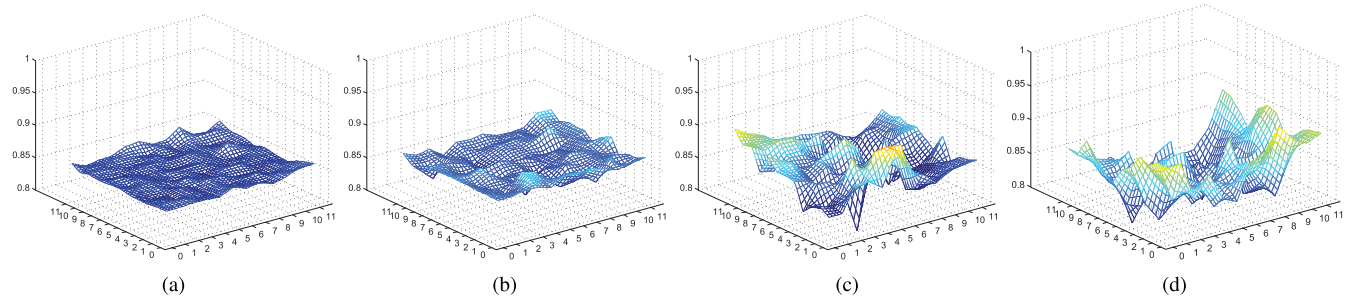

Fig. 11. Effect of reliability-aware power management approach on cores' reliability after 2 years of system activity (TSP-based approach). (a) RA-MOC. (b) RA-MOC-no-balancing. (c) MOC. (d) PGCapping.

reliability balancing (i.e., without Algorithm 6), namely, *RA-MOC-no-balancing* and the full-fledged approach, namely, *RA-MOC*. Using these two steps, we aim at better demonstrating the contribution of reliability-aware power allocation and balancing technique in balancing the overall reliability on the chip. Moreover, we compared RA-MOC and RA-MOC-no-balancing with two state-of-the-art approaches: the original power management strategy without considering reliability [12], called *MOC*, and *PGCapping* [11], which also features lifetime optimization.

Figs. 8 and 9 compare the reliability curves of the three approaches while using TDP and TSP as the maximum power upper bound, respectively. Each graph reports the minimum, the maximum, and the average reliability values of the various cores within the architecture over 6 years of activity. When

TABLE I
SYSTEM LIFETIME IN TERMS OF MTTF

	Using TDP	Using TSP
RA-MOC	7.4 years	6.6 years
RA-MOC without RB	6.2 years	6 years
MOC	6 years	5.4 years
PGCapping [11]	6.1 years	5.7 years

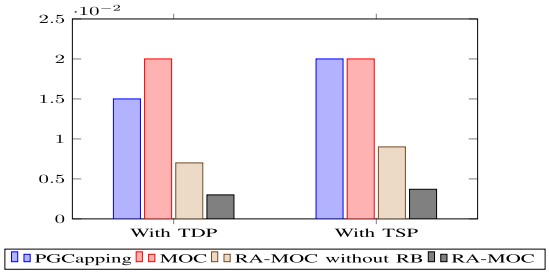

Fig. 12. Standard deviation of reliability for different DPMs after 2 years of system activity.

comparing with MOC and PCGating, the proposed approach minimizes the variance in the reliability values, thus maximizing the average lifetime of the various tiles. Instead, since the two state-of-the-art approaches are reliability agnostic, they distribute the applications without considering the aging values and therefore lead to an unbalanced distribution of the workload and, consequently, of the aging on the cores. This will lead to a lower reliability of some cores that probabilistically will fail earlier. All these considerations are also confirmed by Table I, which reports the mean-time to failure (MTTF) of the system computed according to the obtained reliability curves. RA-MOC is able to obtain an improvement in MTTF of around 23% and 22% for TDP and TSP-based approaches with respect to the reliability-agnostic approach. Finally, if we consider RA-MOC and RA-MOC-no-balancing, it is possible to note that both the two techniques (reliability-aware power allocation and reliability balancing) give a contribution on the prolonging of the lifetime. RA-MOC-no-balancing presents better results than the reliability-agnostic MOC; then the reliability balancing technique in RA-MOC even more improves such results. From this comparison, it is possible to conclude that the proposed approach is able to balance the reliability of the cores and increase the lifetime with negligible performance penalty comparing with the state-of-the-art reliability-agnostic solution (i.e., MOC). In fact, MOC selects regions to apply DVFS only by means of performance-centric metrics. In contrast, RA-MOC performs an accurate selection by also taking into account the aging status.

In order to better appreciate the capabilities of the proposed approach, we take a snapshot of the reliability status of the system after 2 years presented in Figs. 10 and 11 for TDP and TSP, respectively. As can be seen, the reliability distribution obtained by RA-MOC is more evenly distributed compared with those by the other scenarios. The worst case is when no reliability consideration is applied. Moreover, it is interesting to observe that cores age faster while using TSP as the power bound compared with cores while using TDP. That is because of the fact that while using TSP, there exists much more utilization compared to while using TDP, which results in more power consumption and overall temperature and stresses the cores faster. As a final note, Fig. 12 reports the standard deviation regarding the reliability distribution for the aforementioned scenarios in Figs. 10 and 11.

VII. Conclusion

In this paper, we presented a multiobjective feedback controller approach to manage the power budget among the various processing elements of a many-core system running a highly variable workload. The feedbacks to the controller are the processing tiles' power–performance measurements, the tiles aging status, application workloads, and network congestion. Comparing the total system power with the maximum power budget, the controller efficiently changes the voltage and frequency of appropriate tiles to optimize performance while prolonging the lifetime of the system by avoiding stress and thermal hotspots. The results showed enhanced system's throughput, TDP/TSP violation, and overall lifetime for the proposed platform compared with state-of-the-art power management policies. Future directions will consider the integration of more advanced power management policies also acting on the communication subsystem and reliability-aware mapping strategies to better prolong lifetime while not incurring much performance penalty.

References

[1] N. Goulding-Hotta *et al.*, "The GreenDroid mobile application processor: An architecture for silicon's dark future," *IEEE Micro*, vol. 31, no. 2, pp. 86–95, Mar./Apr. 2011.

[2] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," *IEEE Micro*, vol. 32, no. 3, pp. 122–134, May 2012.

[3] A. M. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, and H. Tenhunen, *The Dark Side of Silicon*, 1st ed. Springer, Switzerland, 2016.

[4] ITRS, *International Technology Roadmap for Semiconductors*, accessed on May 27, 2016. [Online]. Available: http://www.itrs2.net/.

[5] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, and L. Shang, "System-level reliability modeling for MPSoCs," in *Proc. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES)*, 2010, pp. 297–306.

[6] L. Wang and K. Skadron, "Implications of the power wall: Dim cores and reconfigurable logic," *IEEE Micro*, vol. 33, no. 5, pp. 40–48, Sep. 2013.

[7] S. Pagani *et al.*, "TSP: Thermal safe power: Efficient power budgeting for many-core systems in dark silicon," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES)*, 2014, Art. no. 10.

[8] P. Bogdan, R. Marculescu, and S. Jain, "Dynamic power management for multidomain system-on-chip platforms: An optimal control approach," *ACM Trans. Design Autom. Electron. Syst.*, vol. 18, no. 4, 2013, Art. no. 46.

[9] R. David, P. Bogdan, R. Marculescu, and U. Ogras, "Dynamic power management of voltage-frequency island partitioned networks-on-chip using intel sing-chip cloud computer," in *Proc. Int. Symp. Netw.-Chip (NOCS)*, 2011, pp. 257–258.

[10] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical power management for asymmetric multi-core in dark silicon era," in *Proc. Design Autom. Conf. (DAC)*, 2013, pp. 1–9.

[11] K. Ma and X. Wang, "PGCapping: Exploiting power gating for power capping and core lifetime balancing in CMPs," in *Proc. Int. Conf. Parallel Architect. Compil. Techn. (PACT)*, 2012, pp. 13–22.

[12] A.-M. Rahmani *et al.*, "Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2015, pp. 219–224.

[13] M. U. K. Khan, M. Shafique, and J. Henkel, "Power-efficient accelerator allocation in adaptive dark silicon many-core systems," in *Proc. Int. Conf. Design, Autom. Test Eur. (DATE)*, 2015, pp. 916–919.

[14] M. U. K. Khan, M. Shafique, and J. Henkel, "Hierarchical power budgeting for dark silicon chips," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2015, pp. 213–218.

[15] M. Shafique, B. Vogel, and J. Henkel, "Self-adaptive hybrid dynamic power management for many-core systems," in *Proc. Int. Conf. Design, Autom. Test Eur. (DATE)*, 2013, pp. 51–56.

[16] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *Proc. Int. Conf. Design, Autom. Test Eur. (DATE)*, 2015, pp. 1521–1526.

[17] M.-H. Haghbayan *et al.*, "Dark silicon aware power management for manycore systems under dynamic workloads," in *Proc. Int. Conf. Comput. Design (ICCD)*, 2014, pp. 509–512.

[18] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. S. Rosing, "Dynamic variability management in mobile multicore processors under lifetime constraints," in *Proc. Int. Conf. Comput. Design (ICCD)*, 2014, pp. 448–455.

[19] T. Kim, B. Zheng, H.-B. Chen, Q. Zhu, V. Sukharev, and S. X.-D. Tan, "Lifetime optimization for real-time embedded systems considering electromigration effects," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2014, pp. 434–439.

[20] J. Howard *et al.*, "A 48-core IA-32 message-passing processor with DVFS in 45 nm CMOS," in *Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, 2010, pp. 108–109.

[21] Kalray. *Kalray MPPA Manycore*, accessed on May 27, 2016. [Online]. Available: http://www.kalrayinc.com/

[22] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Smart hill climbing for agile dynamic mapping in many-core systems," in *Proc. Design Autom. Conf. (DAC)*, 2013, pp. 1–6.

[23] K. K.-W. Chang, R. Ausavarungnirun, C. Fallin, and O. Mutlu, "HAT: Heterogeneous adaptive throttling for on-chip networks," in *Proc. Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, 2012, pp. 9–18.

[24] A. Y. Weldezion *et al.*, "Scalability of network-on-chip communication architecture for 3-D meshes," in *Proc. Int. Symp. Netw.-Chip (NOCS)*, 2009, pp. 114–123.

[25] J. Blome, S. Feng, S. Gupta, and S. Mahlke, "Self-calibrating online wearout detection," in *Proc. Int. Symp. Microarchitecture (MICRO)*, 2007, pp. 109–122.

[26] *Failure Mechanisms and Models for Semiconductor Devices*, document JEP122G, JEDEC Solid State Tech. Association, 2010.

[27] C. Bolchini, M. Carminati, M. Gribaudo, and A. Miele, "A lightweight and open-source framework for the lifetime estimation of multicore systems," in *Proc. Int. Conf. Comput. Design (ICCD)*, 2014, pp. 166–172.

[28] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.

[29] O. Bilgir, M. Martonosi, and Q. Wu, "Exploring the potential of CMP core count management on data center energy savings," in *Proc. Workshop Energy Efficient Design*, 2011, pp. 1–7.

[30] C. Silvano, G. Palermo, S. Xydis, and I. Stamelakos, "Voltage island management in near threshold manycore architectures to mitigate dark silicon," in *Proc. Int. Conf. Design, Autom. Test Eur. (DATE)*, 2014, pp. 1–6.

[31] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *Proc. Int. Conf. Appl.-Specific Syst., Archit. Process. (ASAP)*, 2015, pp. 162–163.

[32] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. Int. Symp. Microarchitecture (MICRO)*, 2009, pp. 469–480.

[33] L. Wang and K. Skadron, "Dark vs. dim silicon and near-threshold computing extended results," Dept. Comput. Sci., Univ. Virginia, Charlottesville, VA, USA, Tech. Rep. TR-2013-01, 2012.

[34] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, pp. 94–125, Mar. 2004.

[35] *TGG: Task Graph Generator*, accessed on Apr. 4,2013. [Online]. Available: http://sourceforge.net/projects/taskgraphgen/

[36] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, and J. Plosila, "CoNA: Dynamic application mapping for congestion reduction in many-core systems," in *Proc. Int. Conf. Comput. Design (ICCD)*, 2012, pp. 364–370.

Amir M. Rahmani (M'08) received the M.Sc. degree from the University of Tehran, Tehran, Iran, in 2009, the Ph.D. degree from the Department of Information Technology, University of Turku, Turku, Finland, in 2012, and the M.B.A. degree jointly from the Turku School of Economics, Belgium, University of Turku, Belgium and European Institute of Innovation and Technology ICT Labs in 2014.

He is currently a University Teacher with the University of Turku. He is co-leading three Academy of Finland projects entitled MANAGE, InterSys, and SPA. His current research interests include energy-efficient and dependable computing, parallel and distributed systems, and Internet-of-Things.

Mohammad-Hashem Haghbayan (S'14) received the B.A. degree in computer engineering from the Ferdowsi University of Mashhad, Mashhad, Iran, and the M.S. degree in computer architecture from the University of Tehran, Tehran, Iran. He is currently pursuing the Ph.D. degree with the University of Turku, Turku, Finland. His current research interests include high-performance energy-efficient architectures, power management techniques, and online/offline testing.

Antonio Miele (M'12) received the M.Sc. degree in computer science from the University of Illinois at Chicago, Chicago, IL, USA, in 2006, and the M.S. degree in computer engineering and the Ph.D. degree in information technology from the Politecnico di Milano, Milan, Italy, in 2006 and 2010, respectively.

He is an Assistant Professor with the Politecnico di Milano since 2014. His research interests include the design of dependable embedded systems, run-time resource management in multi-/many-core systems and reconfigurable systems.

Pasi Liljeberg (M'09) received the M.Sc. and Ph.D. degrees in electronics and information technology from the University of Turku, Turku, Finland, in 1999 and 2005, respectively.

He was an Academy of Finland Researcher from 2007 to 2009. He is currently the Leader of the Internet-of-Things for Healthcare Research Group. He is currently an Adjunct Professor of Embedded Computing Architectures with the Embedded Computer Systems Laboratory, University of Turku.

Axel Jantsch (M'97) received the Dipl.-Ing and Dr.Tech. degrees from the Vienna University of Technology, Vienna, Austria.

He was a Professor of Electronic System Design with the Royal Institute of Technology, Stockholm, Sweden. He is currently a Professor of System-on-Chip with the Vienna University of Technology. His current research interests include VLSI design and synthesis, system-level specification, modeling and validation, HW/SW co-design and co-syntheses, reconfigurable computing, and networks-on-chip.

Hannu Tenhunen (M'83) received the Diploma degree from the Helsinki University of Technology, Espoo, Finland, in 1982, and the Ph.D. degree from Cornell University, Ithaca, NY, USA, in 1986.

He joined the Signal Processing Laboratory, Tampere University of Technology, Tampere, Finland, as an Associate Professor in 1985, where he later served as a Professor and the Department Director. Since 1992, he has been a Professor with the KTH Royal Institute of Technology, Stockholm, Sweden, where he also served as the Dean. He has authored over 600 publications and 16 patents.